Kevin L. Mills, National Bureau of Standards, Washington, D. C.

Testing OSI protocols: NBS advances the state of the art

In addition to other techniques, the National Bureau of Standards uses automatic programming to test protocol implementations.

The drive to develop standard communication protocols for Open Systems Interconnection (OSI) has progressed far beyond the dream phase. Already, working consensus protocols are available for the first four layers of the OSI reference model. These protocols will soon be followed by standard Internet, Session, and File Transfer protocols. In large part, the success of this standardization is due to government-supported laboratory programs allowing trial development and testing of the protocols. The results of the prototype development and testing programs are fed back into the standards development process, thus yielding more mature and realistic standards for protocols.

One of the most effective government programs toward this end is being pursued by the Institute for Computer Sciences and Technology (ICST) at the National Bureau of Standards (NBS). ICST has developed a protocol testing methodology and a set of automated tools that facilitate rapid and effective evaluation of emerging standard protocols. Development efforts of Federal Information Processing Standards (FIPS), American National Standards, and International Standards have all been influenced by the results of these testing programs.

Synopsis

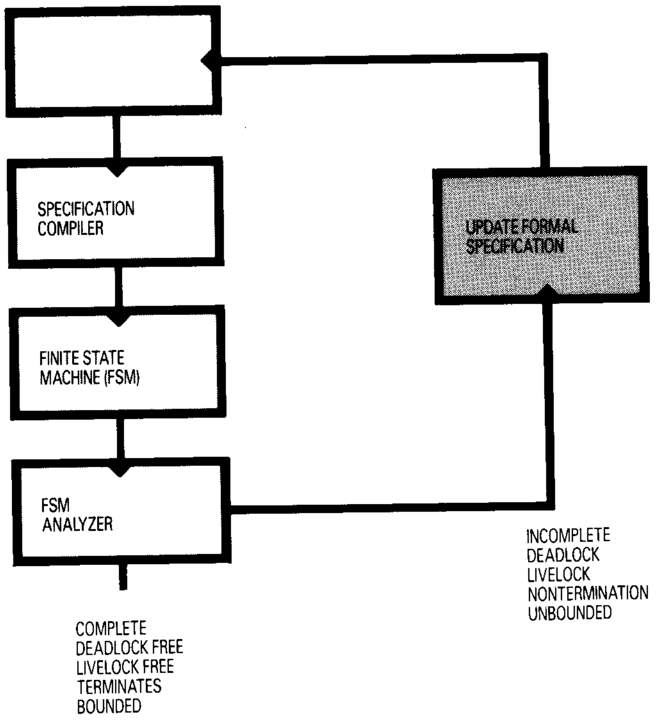
Before the ICST testing methodology can be employed, the proposed protocol standard must be described in a precise notation. The description, called the formal specification, must then be verified as to the services provided and mechanisms used. From the verified specification, a reference implementation is constructed using automated tools in order to reduce the amount of hand-coding. This reference implementation serves as the foundation for a testing architecture that is supported by several automated test tools. This architecture also consists of formally defined testing procedures based on scenarios of protocol service primitives.

Figure 1 illustrates ICST's process for verifying protocol specifications. First, a formal specification of the protocol is created in accordance with ICST rules. The formal specification, which is both machine and human readable, is the input to a specification compiler. This compiler produces a finite state machine (FSM) model of the specification's protocol mechanisms. Then, an FSM analyzer is invoked that examines various subsets of the protocol mechanisms. The FSM analyzer can determine if the protocol described by the formal specification meets five criteria:

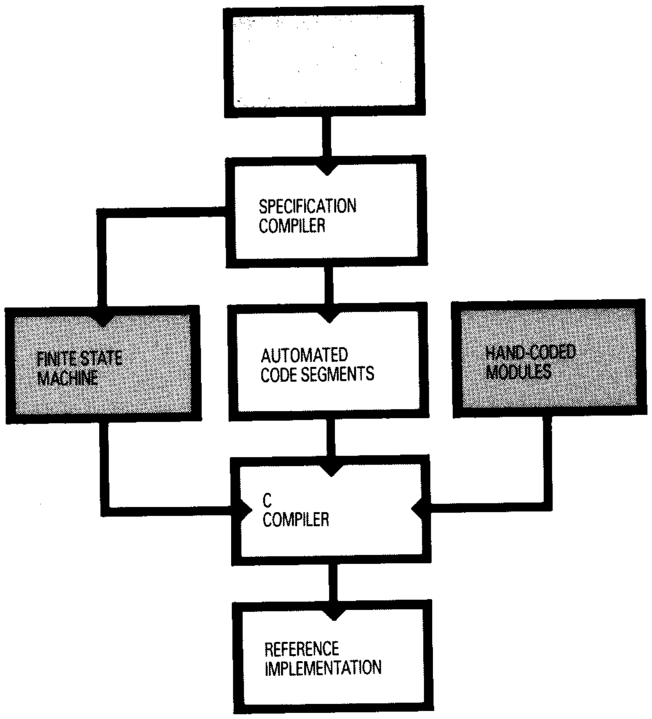
- Completeness: An FSM transition exists for every possible input.
- Free of deadlocks: No possible message sequence will cause the computers to wait eternally for each other.
- Free of livelocks: No possible message sequence exists that might cause the computers to transmit messages forever without making progress on user information. An example of livelock is one machine repeatedly sending the same message while the other always returns an acknowledgment.
- Terminates: For any possible sequence, the machines will conclude in an expected state.
- Bounded: There is no message sequence that cannot be processed. In other words, it is impossible for a computer to build an infinite queue of tasks.

If the FSM analyzer identifies protocol design errors, the formal specification is updated and checked again. After the analyzer verifies the protocol design in terms of the five properties, the formal specification becomes the basis for constructing a reference implementation

1. Five properties. *The finite state machine (FSM), an an established computer modeling tool, is used to test that the specification contains crucial properties.*



2. Error-free. *When possible, automated tools are employed to decrease the chance of human error. Sometimes, however, hand-coded modules are required.*



of the protocol.

A reference implementation must behave correctly with respect to the services and mechanisms defined by its formal specification. In general, it is difficult to ensure that any software implementation of a protocol reflects the original specifications. Fortunately, the ICST process for constructing a reference implementation provides a reasonable assurance that the implementation faithfully reflects the specification. (Fig. 2)

A specification compiler accepts the formal specification as input and produes high-level language code implementing the protocol. This machine- independent code represents correct protocol behavior as a set of finite state machine transitions and simultaneous actions. Placing the code into a specific operating system environment requires the addition of hand-coded, machine-dependent modules. For the most part, these hand-coded modules do not affect correct protocol behavior, although there are exceptions. These include a small number of machine-dependent primitive operations identified in the protocol specification (for example, selecting sequence space parameters or finding a network address).

For a reference implementation of the NBS Class 4 Transport Protocol, the automatically generated code contains 4,500 lines of C language code; the hand-coded modules contain 5,700 lines. Thus, approximately 40 percent of the reference implementation for Class 4 Transport was automatically generated from the specification. Since much of the reference implementation is derived without the chance for human error, the behavior of the protocol has a high probability of correctly reflecting the specification. Because it generates the protocol correctly, the reference implementation serves as the foundation for the ICST testing architecture.

Structure

Figure 3 shows the architecture of ICST's facility for testing OSI protocols. The architecture consists of a test center host and a remote client host attached on the same network. (The remote client is the government or industry organization that wants its implementation tested.) The ICST provides a set of files, containing test steps (scenario files), that are executed at each host. A scenario interpreter at each host reads the scenario files and provides an interface to the OSI protocol layer being tested. The client implements its interpreter from a functional specification provided by the ICST.

An enhanced scenario interpreter at the test center provides the functions specified for the client scenario. This interpreter records summary results for each scenario and logs both the results of each test step and all the data received from the client.

The test center interpreter provides an interface to the reference implementation for the tested protocol layer. Since the reference implementation only produces a correct protocol, it cannot indicate the remote implementation's behavior in the face of protocol perturbations. In order to analyze this behavior, ICST developed a program called an exception generator that is controlled by the test center interpreter. The generator induces errors in the protocol data units, that is, the packets containing protocol control information and, optionally, data.

This exception generator resides between the reference implementation and the layer below in order to receive correct PDUs and to induce errors. Under control of the test center scenario interpreter, the exception generator simulates network duplication, loss, misordering, and damaging of PDUs. The interpeter can change any field of a PDU and can add, delete, and modify fields within PDU headers, including checksums. The exception generator can operate on PDUs both transmitted and received by the reference implementation.

All of the reference implentation's incoming and outgoing protocol traffic can be passed through the exception generator. Consequently, the generator can keep a complete record of all PDUs exchanged between the reference and remote implementations. This log also reflects all errors induced by the exception generator, including lost, misordered, duplicated, and damaged PUDs.

An additional feature of the ICST test architecture is the ability to loop back at multiple levels. For example, when testing Class 4 Transport, loop-back can be activated at the transport level, the exception generator level, and the network level. In fact, before it began testing with remote clients, ICST used this feature to check all of its components in isolation.

More aids

In addition to those already described, the ICST has implemented three other protocol testing tools. One of them, the automated scenario generator, produces scenario files in a grammar derived semi-automatically from the formal protocol specification. This grammar is the set of rules for generating all combinations of valid commands for the scenario interpreter. Since both the grammar and the scenario files are produced automatically, no meaningful test scenarios will be overlooked.

A PDU log analyzer provides a human readable representation of PDU exchanges logged by the exception generator. A hexadecimal representation is always provided, and when feasible, the PDU fields are decoded into their mnemonics.

A performance monitor interprets PDU sequences and can categorize PDUs by type, length, and other parameters. In addition, it allows performance analysis. The monitor operates from the PDU stream logged by

3. Total control. *Through the exception generator, the test center scenario interpreter induces PDU errors in a controlled manner. The test center's PDU log file can record all PDUs for on-line or off-line analysis. The same architecture will be used when higher level standards solidify and are implemented.*

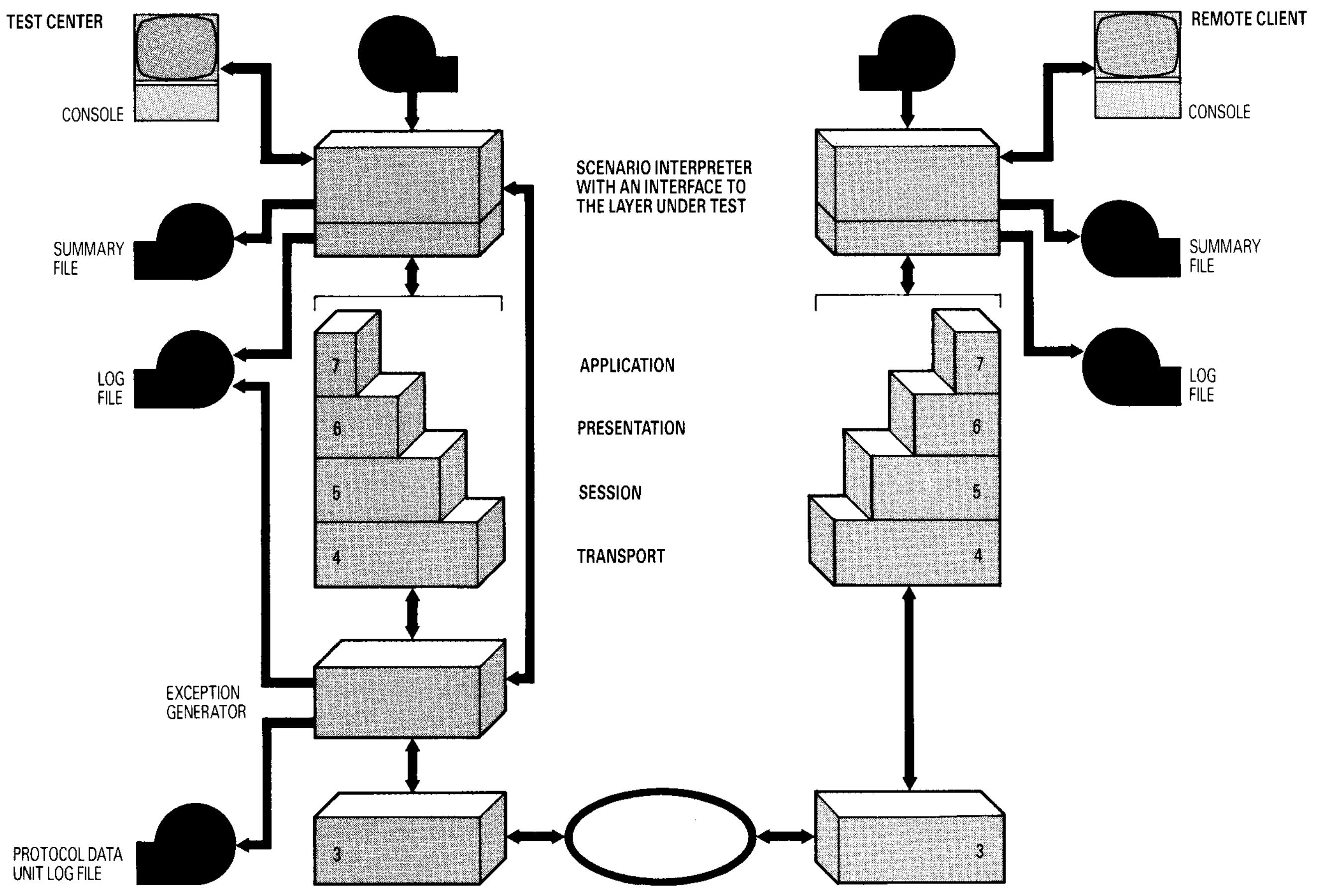


Table 1: Summary of ICST protocol test tools

TOOL	PURPOSE
SPECIFICATION COMPILER	BUILDS A FINITE STATE MACHINE (FSM) REPRESENTING PROTOCOL MECHANISMS GENERATES THE PROGRAM SEGMENTS THAT IMPLEMENT THE ACTIONS ASSOCIATED WITH FSM TRANSITIONS
FSM ANALYZER	VERIFIES PROTOCOL SPECIFICATION FSM WITH RESPECT TO COMPLETENESS, FREEDOM FROM DEADLOCK AND LIVELOCK, TERMINATION, AND WHETHER IT IS BOUNDED
REFERENCE IMPLEMENTATION	PROVIDES CORRECT IMPLEMENTATION OF PROTOCOL BEHAVIOR IN ACCORD WITH THE PROTOCOL SPECIFICATION
SCENARIO INTERPRETER	INTERPRETS TEST SCENARIOS AND DRIVES THE REFERENCE IMPLEMENTATION AND EXCEPTION GENERATOR AS REQUIRED GENERATES/COMPARES DATA FOR A TEST RECORDS TEST RESULTS
EXCEPTION GENERATOR	INDUCES PROTOCOL PERTURBATIONS AS DIRECTED BY THE SCENARIO INTERPRETER LOGS PDU EXCHANGES PROVIDES PDU EXCHANGES IN REAL-TIME TO THE PERFORMANCE MONITOR
PERFORMANCE MONITOR	CONSUMES PDU STREAM AND PRODUCES PROTOCOL PERFORMANCE MEASURES
SCENARIO GENERATOR	CREATES SCENARIOS BY EXAMINING A COMPOSITE SERVICE GRAMMAR
PDU LOG ANALYZER	PRODUCES A HUMAN-READABLE REPRESENTATION OF A BINARY ENCODED PDU STREAM

the exception generator. Performance measurements can be done off-line using the PDU log file or in real-time with PDUs passed from the exception generator. In addition, when a local network is used for testing, the performance monitor can be configured as an independent network node providing measures across network hosts. (See Table 1)

Hand in hand

An ICST protocol test is completely defined by a pair of complementary test scenarios. Each test scenario is a set of commands representing protocol service primitives with parameters and, optionally, exception generator commands. Service primitives possible within the test scenarios for Class 4 Transport are shown in Table A, B.

During a test, one scenario is executed at the test center and a complementary scenario is executed at the remote client site. Active service primitives direct a scenario interpreter to request a protocol service. Simultaneously, passive service primitives direct a scenario interpreter to expect an indication that an action will occur. To test connection establishment, for example, the test center scenario would have a CONNECT REQUEST service primitive, and the remote client scenario would contain a CONNECT INDICATION service primitive.

The ICST has identified 421 test scenarios for testing of Class 4 Transport implementations. Sixty of them were selected from a larger number of automatically generated scenarios, and the rest were generated by hand. The table shows the categories of test scenarios

A: Test scenario service primitives

ACTIVE PRIMITIVE	COMPLEMENTARY PASSIVE PRIMITIVE
CONNECT REQUEST	CONNECT INDICATION
CONNECT RESPONSE	CONNECT CONFIRM
DATA REQUEST	DATA INDICATION
EXPEDITED REQUEST	EXPEDITED INDICATION
CLOSE REQUEST	CLOSE INDICATION
DISCONNECT REQUEST	DISCONNECT INDICATION

B: Test scenarios defined by ICST

CATEGORY	NUMBER OF SCENARIOS
VALID SEQUENCES OF SERVICE PRIMITIVES	64
INVALID SEQUENCES OF SERVICE PRIMITIVES	42
PARAMETER VARIATIONS ON SERVICE PRIMITIVES	47
INVALID SEQUENCES OF PDUs	107
PARAMETER VARIATIONS ON PUDs	76
PROTOCOL MECHANISM SPECIFIC TESTS	25
AUTOMATICALLY GENERATED TESTS	60

defined and the number in each category. These include 361 hand-generated scenarios and 60 scenarios selected from a large number of automatically generated scenarios.

Procedures

Prior to the start of testing, the test scenarios to be executed at the remote client sites are distributed on magnetic tape. Testing is then conducted according to a plan agreed upon by personnel at the test center and the remote client site. ICST's general plan is to perform test scenarios in a building block fashion, beginning with the hand-generated scenarios and ending with the automatically generated scenarios.

The first set of tests ensures that the remote implementation is able to establish and terminate connections properly. Next are tests for simplex data flow in each direction. Similar tests are conducted for expedited (high priority) data flow. After testing simplex data flow in both directions, more complicated sets of scenarios are used to test full-duplex flow of normal and expedited data.

After solid operation is assured under a mix of normal situations, tests are executed that use the controlled introduction of protocol perturbations. Finally, automatically generated tests are invoked.

For the ICST reference implementation of Class 4 Transport, all scenarios have been executed in loopback mode to ensure that the reference implemenation operates as required. During this test period, reference implementation errors were discovered, corrected, and retested.

Using the method

The methodology and tools described above have also been used to construct and test the NBS Class 2 Transport Protocol reference implementation and an X.25 network interface sublayer. For Class 4 Transport, the connection management phases of the formal protocol specification have been verified. ICST's testing procedure has been used to test a single computer with loop-back at several levels and has been employed between computers at the ICST and Bolt, Beranek, and Newman in Cambridge, Massachusetts, over the ARPANET and Telenet networks. The degree of completeness of the testing methodology can best be indicated by the scope of errors uncovered while testing Class 4 Transport. The types of errors discovered during this testing include: protocol design, resource management, transport user interface, coding, documentation, PDU encoding/decoding, and even errors in the host operating system. The rigor and scope of these tests has greatly increased the practicality and correctness of the proposed NBS Transport Protocol FIPS.

Future plans

In 1984, ICST will continue a 1983 program of cooperative testing with commercial and government implementors of the NBS Class 2 and Class 4 Transport Protocols. These tests will operate on a variety of local networks, satellite links, and Arpanet and Telenet. Through this program, ICST will be able to assist commercial and government implementors in buiding correct transport protocol implementations. In addition, it will also be able to assess and improve protocol performance on specific network types. Lastly, the program will allow ICST to continue to advance the state of the art in development and testing of OSI protocols. The results of the research programs, it is hoped, will assist governmental, national, and international standards bodies in their work on both emerging and evolving protocol standards.

Additional reading:

1. J. Stephen Nightingale. "Protocol Testing Using a Reference Implementation." Proceedings of the IFIP WG 6.1 Second International Workshop on Protocol Specification, Testing, and Verification, May, 1982.
2. R. J. Linn and J. S. Nightingale. "Some Experience with Testing Tools for Testing OSI Protocol." Proceedings of the IFIP 6.1 Third International; Workshop, On Protocol Specification, Testing, and Verification, May 1983.
3. R. J. Linn and W. H. McCoy. "Producing Tests for Implementations of OSI Protocols." Proceedings of the IFIP 6.1 Third International Workshop on Protocol Specification, Testing, and Verification, May 1983.
4. T. P. Blumer and R. L. Tenney. "A Formal Specification Technique and Implementation Method for Protocols." Computer Networks, Volume 6, Number 3, 1982. ■